VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Démon pro sledování síťových prvků s výstupem do Nagiosu

Daemon for monitoring network devices with output to Nagios system

2011 Jan Bednář

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 5. května 2011 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2011 .....

Rád bych na tomto místě poděkovala všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Bakalářská práce se zabývá problémem monitorování síťových prvků, převážně přepínačů. Popisuje vytvoření nástroje pro monitorování těchto prvků, který umožňuje získávání důležitých informací prostřednictvím protokolu SNMP, jejich vyhodnocování a předávání dál, pomocí protokolu NSCA systému Nagios, který již přímo tyto data zpracuje. Práce se postupně zabývá analýzou dostupných technologií, řešením problémů vyplývajících z těchto technologií a poté zmiňuje detaily implementace jednotlivých částí.

Klíčová slova: SNMP, Python, Nagios, NSCA

Abstract

Bachelor work deals with problems in monitoring of network devices, mainly switches. Describes creation of monitoring tool for these devices, which enables gathering of important informations through SNMP protocols, their evaulation and passing by through NSCA protocol to Nagios system for processing. Work progressively deals with analysis of usable technologies, solving problems with using these technologies and talks about details of each part implementation.

Keywords: SNMP, Python, Nagios, NSCA

Seznam použitých zkratek a symbolů

CPU	–	Central Processing Unite - hlavní řídící jednotka počítače
CRC	–	Cyclic Redundancy Check - cyklicý redundantní součet
FTP	–	File Transfer Protocol - protoko pro přenos souborů
GPL	–	General Public License - licence pro svobodný software, dovolující úpravu a šíření zdrojových kódů
GUI	–	Graphical User Interface - grafické uživatelské prostředí
HTTP	–	Hyper-Text Transfer Protocol - protokol pro přístup k webu
ICQ	–	I Seek You - program pro zasílání online zpráv mezi u uživateli
IP	–	Internet Protocol - protokol pro přenos dat v síti
MIB	–	Management Information Base
NRPE	–	Nagios Remote Plugin Executor - služba spouštíci vzálené testy
NSCA	–	Nagios Service Check Acceptor - služba pro pasivní testování
OID	–	Object Identifier - Identifikátor SNMP objektu
PDU	–	Protocol Data Unit - typ SNMP dotazu
POP3	–	Post Office Protocol verze 3 - přístup k elektronické poště
SMS	–	Short Message Service - krátká textová zpráva
SMTP	–	Simple Mail Transfer Protocol - protokol určený k zasílání elektronické pošty
SNMP	–	Simple Network Monitoring Protocol
SVN	–	Subversion - systém pro správu a verzování zdrojových kódů
UDP	–	User Datagram Protocol - protokol bez záruky přenesení dat

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešním moderním světě si člověk již neumí představit život bez různých elektronických komunikačních technologií. Pomocí těchto technologií přistupuje dnes a denně k drtivé většině informací, které má. Student si dnes nedokáže ani představit život a studium bez internetu. Banky pomocí internetu převádí všemožné finanční částky. Úřady a firmy komunikují pomocí svých informačních systémů a elektronické pošty, skrze veřejné i soukromé komunikační kanály. Všechna odvětví průmyslu, obchodu i běžného života jsou neodmyslytelně provázány různými typy počítačových sítí.

Běžný uživatel se však zajímá pouze o to, aby se k informacím dostal. V opačném případě je značně nespokojen, proto se snažíme takovým situacím předejít. Všechny počítačové sítě jsou propojeny kabely, které, dříve či pozdeji, končí v nějakém přepínači, případně jiném aktivním prvku. Pokud je přes takovýto prvek připojeno více počítačů, které jsou pro některé skupiny lidí důležité, je třeba, aby provoz přes tyto uzly byl monitorován a případné chyby byly odhaleny v co nejkratší možné době.

Cílem této bakalářské práce vytvoření programu, který umožní informace z těchto uzlů hromadně sbírat a odesílat na server, který monitoruje ostatní části a aspekty sítě.

První část se zabývá vysvětlením teorie a principů, pomocí kterých se komunikuje. Je zmíněn skriptovací jazyk Python a jeho základní popis. Významnou část tvoří popios SNMP komunikace a jeho objektů. Jejich pochopení je důležité pro poruzumnění zbytku práce.

Druhá část řeší problematiku vzniklou z požadovaných technologií. Bohužel v průběhu řešení práce došlo k nalezení několika navzájem nekompatibilních verzí jednotlivých prostřední a knihoven. Popis těchto problémů a jejich řešení by měl poskytnout funkční základ pro další práce, řešící podobná úskalí.

Třetí část je věnována popisu programu a jeho jednotlivých komponent. Detailněji ukazuje struktůru programu a vysvětluje použití jednotlivých knihoven.

2 Teoretický rozbor

Tato bakalářská práce byla vyvíjena pro firmu LinuxBox.cz, s.r.o., která měla následující požadavky:

- démon pro sběr dat bude naprogramován ve skriptovacím jazyce Python
- informace z přepínačů budou načítána pomocí protokolu SNMP
- data budou zasílána pro další zpracování systému Nagios pomocí protokolu NSCA
- bude vytvořen nástroj pro generování definic nových zařízení do systému Nagios
- aplikace bude jako funkční celek provozuschopná pod systémem GNU/Linux

Primárním cílem je nahradit současný interní nástroj `lxmon`, naprogramovaný před několika lety touto firmou v jazyce Perl a sloužící pro monitorování síťových zařízení v systému Big Sister (http://www.bigsister.ch/).

Tomuto účelu bude také přizpůsobeno vyhodnocení výsledků testů a jejich výstupní formátování do požadovaného tvaru. Bude pracováno pouze s výběrem relevantních hodnot, důležitých pro získání informací o stavu rozhraní.

2.1 Python

Jazyk Python začal vznikat v roce 1989 ve výzkumném ústavu v Amsterodamu. Při jeho zrodu stál Guido van Rossum a je vidět, že u návrhu dostatečně přemýšlel. Vznikl promyšlený jazyk, který je stále ve vývoji. Jméno dostal podle pořadu BBC Monty Python's Flying circus. V současné době běží na mnoha platformách (Linux, Win, Mac, WinCE, OS2, Java). Stejně tak programy v něm napsané lze na těchto systémech téměř vždy spouštět bez úprav.

A jaký Python vlastně je? Čistý objektový jazyk se správou výjimek, kompilací do bytecodu, mnoha vysokoúrovňovými typy (řetězce, seznam, asociativní pole), plně podporující Unicode, lze jej doplnit o vlastní vestavěné typy a funkce pomocí C/C++, nebo naopak lze interpret začlenit do programu v jiném jazyce. Základní balík obsahuje velké množství modulů, které lze ihned používat ve vašem programu. Jmenujme moduly pro přístup k databázím, GUI, službám operačního systému, HTTP, FTP, POP3, SMTP a mnohým jiným protokolům. Samozřejmostí jsou regulární výrazy. Definuje také několik modulů pro přístup k vnitřním mechanismům Pythonu (garbage collector, parser, kompiler). V Pythonu je taktéž napsán debugger a profiler tohoto jazyka. [1]

Dnes je Python nasazen na serverch po celém světě, nejčasteji ve verzích 2.4 a 2.6. Již delší dobu exituje paralelně verze 3.0, která ovšem není zpětně plně kompatibilní a proto je její nasazení do ostrého provozu pradoxně ne příliš populární. Aplikace jsou tak nadále vyvíjeny pro starší, avšak stále vyvíjené větve tohoto jazyka. Jako příklad nám může sloužit oficiální seznam Python hostingových serverů uveřejněný na http://wiki.python.org/moin/PythonHosting. Z tohoto seznamu více než 150 serverů se dozvíme, že přibližně pouze 1/8 hostingů užívá Python 3.

Poznámka 2.1 Pokud chceme z jiných verzí Python (například Python 3) používat některé vylepšení, nabízí se nám knihovna `__future__`, ze které lze naimportovat například použití `print` jako funkce, případně mnou použité `unicode_literals`, pro standardní použití znakové sady unicode. Celý zápis vypadá následovně:

```
#do A i B vlozime stejny text
a="aa"
from __future__ import unicode_literals
b="bb"
#vysledky (a == 'aa', ale b == u'bb')
type(a)
<type 'str'>
type(b)
<type 'unicode'>
```

Výpis 1: Importování unicode znakové sady

2.2 Protokol SNMP

SNMP, neboli Simple Network Management Protocol pracuje nad protokolem UDP a definuje jednoduché komunikační schéma, umožnující rychlé doručení požadavků a odpovědí mezi počítači, na kterých běží aplikace SNMP. Protokol SNMP má na starosti doručování těchto požadavků a odpovědí za uvedené aplikace. Funguje nezávisle na specifikackých funkcích aplikací, architektuře nižších vrstev či aplikacích vyšších vrstev. Díky tomu je SNMP jednoduchým a obecným, přesto však vykonným protokolem pro správu sítě, který je možno portovat na nejrůznější platformy a operační systémy. Protokol SNMP má tři hlavní entity, které dovolují vzdálenou správu sítí:

- **Správce/Manager** - Generátory příkazů a příjemci oznámení
- **Agent** - Odpovídá na příkazy a vytváří oznámení
- **Proxy** - Předává síťový provoz SNMP

2.2.0.1 Správci SNMP SNMP Manager má na starosti vytváření dotazů, jejich zasílání SNMP agentům. Řeší také následný příjem zprávy a příjem zpráv, které nejsou vyžádané správcem, ale jsou důležité (tzv. „trap messages“)

2.2.0.2 Agenti SNMP Agenti přímají požadavky od správců a generují k nim příslušné odpovědi. Agenti mohou také případně zasílat správci hodnoty v určitých časových intervalech, nebo v případě, že nastanou určité situace (např. při nadměrném vytížení CPU, zaplnení disku, atp.).

2.2.0.3 Proxy Proxy pouze předává síťový provoz a v našem případě nás jeho princip ani úloha v síti nemusí zajímat

2.2.1 Komunita SNMP

SNMP ve verzi 1 a 2c využívá tzv. komunitu k oddělení pravomocí jednotlivých správců. Jedná se ve svém podstatě o jakési textové heslo, podle kterého agent pozná, ke kterým záznamům má daný správce přístup, případně, zda-li má povolen i zápis určitých hodnot. Výchozí nastavení definuje komunitu `public`, pro kterou je povoleno pouze čtení. Ve verzi SNMPv3 je tento princip již vyřešen autentizací pomocí zaslání jména a hesla.

2.2.2 Databáze MIB

Každý agent ukládá kolekci svých lokálně řízených objektů v tzv. MIB - Management Information Base. Jedná se o stromově orientovanou databázi všech objektů. Správce musí při pokusu o přístup k údajům definovat, kterou hodnotu požaduje. Každá hodnota v SNMP je jednoznačne identifikována pomocí číselnehé označení OID - Object Identifier. Toto označení je tvořeno posloupností čísel, oddělených tečkou. Jednotlivé čísla specifikují uzly MIB stromu. MIB databáze navíc pro všechny hodnoty specifiguje nejen jejich číselné hodnoty, ale také název.

Příklad 2.1
Příkladem OID může být třeba hodnota 1.3.6.1.2.1.2.2.1.6.1, které odpovídá textová verze z MIB databáze iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifPhysAddress.

■

2.2.3 SNMP dotaz

Možností, jak se dotazovat SNMP agenta na informace je více. V literatuře nejčasteji uváděný je pravděpodbně `get`. Jsou zde ale také další možnosti volání, například pokud chceme zjisit více hodnot najednou.

- **get** - načte jednu hodnotu
- **getnext** - načte další hodnotu, která podle MIB stromu následuje za aktuálním OID
- **getbulk** - načte sekvenci hodnot, které následují za aktuálním OID. Počet hodnot, které se načítají se definuje při requestu
- **getwalk** - projde celý uzel MIB stromu pomocí `getnext`. To znamená, že vytvoří pro každou hodnotu nové spojení
- **getbulkwalk** - načítá celý MIB strom pomocí `getbulk`, který zaručí několikanásobně menší počet spojení.
- **gettable** - získá informace z celého aktuálního MIB stromu a uspořádá je do přehledné tabulky.

Poznámka 2.2 Metoda `gettable` je dostupná pouze z příkazové řádky terminálu a není implementovaná do bindu pro Python, Perl, PHP, ani ostatní jazyky.

2.2.4 Formát paketu

Formát SNMP požadavku a odpovědi má stejnný formát, který obsahuje tyto informace:

- **Version** - verze protokolu (v1, v2c, v3)
- **Community String** - textový název komunity
- **PDU Type** - typ dotazu
  - **0** - GetRequest
  - **1** - GetNextRequest
  - **2** - GetResponse
  - **3** - SetRequest
  - **5** - GetBulkRequest
- **Request ID** - jedinečné číslo dotazu, které je stejné, jako číslo odpovědi, aby bylo možné zjistit, která odpověď patří kterému dotazu
- **Error Status** - Případné číslo chyby
- **Error Index** - pořadové číslo hodnoty v odpovědi, které vykazuje chybu popsanou v Error Status
- **Value(s)** - jeden nebo více OID, které mají být zjištěny, nebo jsou již vráceny agentem.

Version	Community	PDU Type	Request ID	Error Status	Error Index	Value(s)

Tabulka 1: Obecný formát SNMP paketu

Příklad 2.2
Ukázka, jak mohou vypadat SNMP pakety pro požadavek 1.3.6.1.2.1.2.2.1.6.1, zmíněný výše:

v2c (1)	public	0	1234567890	noError (0)	0	1.3.6.1.2.1.2.2.1.6.1: null

Tabulka 2: Příklad SNMP get-request

v2c (1)	public	2	1234567890	noError (0)	0	1.3.6.1.2.1.2.2.1.6.1: 00:11:22:33:44:55

Tabulka 3: Příklad SNMP get-response

[4, 5, 6] ■

2.2.5 Typy objektů

SNMP protokol specifikuje několik typů objektů, které se liší v závislosti na datovém typu a použití.

- **INTEGER** - jednoduché celé číslo. Specifikace neomezuje velikost takového čísla, většina výrobců však používá datového typu Int32
- **COUNTER** - nezáporný integer, který se plynule zvětšuje, až dosáhne maximální hodnoty (2^{32} - 1), poté začíná znovu od nuly. Jak již jméno napovídá, používá se zejména na počítání zajímavých událostí v systému (např. počet přenesených dat, odeslaných e-mailů, atd.). Absolutní hodnota je méně důležitá, než rozdíl (delta) od posledního vzorku, ze kterého lze vyčíst rychlost změn
- **GAUGE** - nezáporný integer. Hodnota Gauge je maximální možná hodnota, kterou může sledovaná informace dosáhnout. Absolutní možná hodnota je opět (2^{32} - 1)
- **TICKS** - nezáporný integer reprezentující v setinách sekundy čas od jisté doby. Může být použit k vyjádření doby chodu nějakého zařízení od jeho zapnutí nebo změny jeho stavu
- **IPADDR** - 32 bitová hodnota IP adresy
- **OCTETSTR** - sekvence bajtů (octets), používáná k vyjádření řetězců. Ve výjmečných případech se užívá pro jiné datové typy, například k zápisu MAC adresy
- **OBJECTID** - reprezentuje pořadové číslo uzlu

2.3 Nagios

Nagios je opensource systém, který se začal vyvíjet pod původním názevem *Netsaint* a v roce 2002 byl finálně přejmenován na Nagios. Je vyvíjený na GNU/Linux a slouží k automatizovanému a sofistikovanému monitorování počítačových sítí a jejich služeb.

Jeho název je akronymem slovního spojení *Nagios Ain't Gonna Insist On Sainthood*, což by se dalo volně přeložit jako *Nagios není připravený stát se svatým*

Program je vydaný pod licencí GNU GPL, tudíž je povoleno jeho použití v soukromé i komerční sféře, je povoleno jeho upravování, kopírování a distribuce.

Samotný Nagios neumí testovat naprosto nic. Pro testování služeb využívá zásuvné moduly, tedy malé prográmky a skripty, které testují určité služby a Nagiosu pouze předávají výsledky testů. [2]

2.3.1 Výsledky testů

Nagios dokáže spracovat pouze 4 hodnoty výsledku jakéhokoli testu. Ty jsou označeny čísly od nuly po trojku.

- **0 = OK** - Plugin dokončil testování služby a podle testu vyhodnotil, že vše pracuje správně

- **1 = Warning** - Plugin dokončil testování služby, ale vyhodnotil výsledky testu jako varovné. Hodnota se porovnávala s administrátorem předem zadanou hodnotou pro Warning
- **2 = Critical** - Plugin dokončil testování služby, ale vyhodnotil výsledky testu jako kritické. Hodnota se porovnávala s administrátorem předem zadanou hodnotou pro Critical. V drtivé většině případů se jedná o situaci, kdy služba není spuštěna, nebo je z výsledku testu patrné, že služba není schopna normálního provozu.
- **3 = Unknown** - Plugin nebyl schopen službu otestovat a podat o ní předpokládané výsledky.

2.3.2 Oznámení

Administrátor má možnost v případě, že některé z testů skončí jiným výsledkem, než je OK, nadefinovat příkaz, který se provede. V praxi se potom nejčastěji používají skripty, které administrátorovi neprodleně odesílají hlášení o stavu. Nejpoužívanější a nejoblíbenější možnosti jsou SMS, emaily, či jiná forma elektronické komunikace, například ICQ (není-li to v rozporu s aktuální ICQ licencí) nebo Jabber. Lze také definovat, že pro různé výsledky testů budou zasílány různé oznámení. Často se tedy využívají emaily pro Warning a SMS pro Critical.

2.3.3 Objekty

Nagios pro definice služeb, zařízení a příkazů používá několik základních objektů. Tyto objekty se spojují do skupin, aby bylo například možné pro všechny objekty stejné skupiny vykonat stejné testy.

- **host** - zařízení, které chceme monitorovat
- **hostgroup** - skupina zařízení typu `host`, které jsou stejného typu. Obvykle se užívají skupiny, které zastřešují prvky typu switch, server, tiskárna, atd.
- **service** - služba, kterou chceme monitorovat. Jedná se buďto o veřejně dostupné služby typu PING, HTTP, FTP, nebo o informace o systému, tedy volné místo na disku, zatížení CPU, počet přihlášených uživatelů, atp.
- **servicegroup** - skupina podobných služeb. Hlavní výhodou je zpřehlednění výsledků.
- **contact** - kontakt, který má být informován o předem definované události. Na tomto místě se také definuje, jakým způsobem má být situace oznámena, tedy který příkaz se má vykonat
- **contactgroup** - skupiny kontaktů, užívané pro zjednodušení konfigurace

- **timeperiod** - časové intervaly, ve kterých má být `contact` informován. Pokud tedy nechceme, aby nás systém rušil v nočních hodinách a víkendech, ať se děje - co se děje, nastavíme to právě zde.
- **command** - příkaz, který se provede, po jeho zavolání, při testech a notifikacích uživatele

Všechny tyto objekty mají vlastní konfigurační soubory, ke kterým je definována cesta v `nagios.cfg` a musí být napsány podle stejné syntaxe.

```
define <objekt> {
    <klic>  <hodnota>
    }
```

2.3.4 Testování

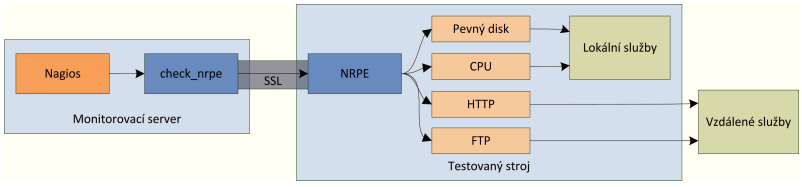
Využívají se dva druhy testů.

1. Aktivní
2. Pasivní

2.3.4.1 Aktivní testování Pro testování se využívá primárně tzv. aktivní testování. Jedná se o testy, které spouští sám Nagios, pomocí definovaných příkazů. Tyto testy se spouští v přesně daný časový okamžik nebo na speciální vyžádání systému a jsou vykonávány skripty nebo programy na straně serveru.

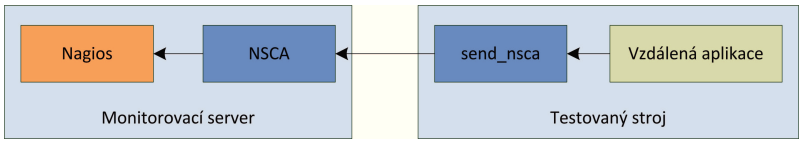
2.3.4.2 Pasivní testy Testování na vzdálených strojích se obyčejně provádí pomocí tzv. pasivních testů. Pro tyto případy běží v rámci Nagiosu dva démoni, kteří se starají o příjem testů, respektive jejich spouštění.

2.3.4.2.1 NRPE - Nagios Remote Plugin Executor Jak již název napovídá, jedná se o službu, které na vzdálených strojích spouští testy a výsledky těchto testů odesílá zpět. Klientská část `nrpe_check` se připojí na server `NRPE`, který je instalován na stroji, který chceme testovat. Následně se pomocí NRPE serveru, vyhodnotí všechny definované testy a následně se výsledky testů odešlou zpět NRPE klientovi, který běží na straně Nagiosu.



Obrázek 1: Diagram testování NRPE

2.3.4.2.2 NSCA - Nagios Service Check Acceptor Pasivní NSCA testování funguje neprosto opačným směrem. Společně s Nagiosem je spuštěn `NSCA` server, který naslouchá a čeká, že mu některý ze vzdálených strojů zašle data. Na vzdáleném stroji běží démon, respektive skript, u kterého je zajištěno spouštění v určité časové intervaly, případně při definované události. Tento skript se pomocí `send_nsca` klienta připojí k NSCA serveru a odešle informace o testovaném stroji a jeho službách.



Obrázek 2: Diagram testování NSCA

NSCA podporuje již v základu přenos dat nejen ve formě plain text, ale umožňuje data šifrovat nejrůznějšími algoritmy, od jednoduchého XORu, až po několikaset-bitové sofistikované šifrovací algoritmy typu SAFER+, jehož mutace se využívá v Bluetooth komunikaci. Paket, který posílá výsledky testu musí mít přesně danou strukturu.

Verze paketu	CRC	Čas odeslání	Návratový kód	Hostname	Služba	Status

Tabulka 4: Formát NSCA paketu

`Návratový kód` je číselný výsledek testu popsaný výše, nabývající hodnot 0 až 3. `Hostname` je jmenné označení testovaného stroje. `Služba` označuje název testované služby a `Status` je textová informace, které doplňuje test a detailněji popisuje výsledek (např. velikost místa na disku, počet přenesených dat, doba odezvy, atd.).

3 Problematika NET-SNMP

Při zadávání této bakalářské práce byl jako požadavek při SNMP komunikaci uveden balík Net-SNMP (http://www.net-snmp.org/), který je třeba pro komunikaci použít. Jedná se o hojně využívaný balík, umožnuje díky portaci užití ve skriptovacích jazycích, jako je Perl, Python, PHP a další. Ve firmě LinuxBox.cz, s.r.o. se ve velké míře používá ve spojení s jazykem Perl, neboť samotný Nagios má veškerou NSCA komunikaci naprogramovanou právě v Perlu.

Pro testování programu a SNMP komunikace mi byl zpřístupněn gigabitový čtřicetiosmi portový přepínač firmy 3Com s podporou vzdálené správy, který má hostname `lbxovasw3` a je dostupný na interní IP adrese `10.76.1.4`.

3.1 Výběr verze Pythonu

3.1.1 Python 2.4

Na všech serverech v ostrém provozu, kde se používá Python, je implicitně nainstalován Python ve verzi 2.4. Byl tedy nainstalován net-snmp balíček pro tuto verzi. Záhy ale bylo zjištěno, že Python ve verzi 2.4 společně s net-snmp není stabilní. Přečtením několika internetových fór bylo jisté, že verze 2.4 není s tímto balíkem kompatibilní.

Python 2.4 si tedy můžeme označit za nekompatibilní a dále se jím nezaobírat

3.1.2 Python 2.6

Zde nastává první okamžik, který tuto práci komplikuje. Je nutné na všechny případné servery zvlášť doinstalovat k Python 2.4, také paralelně verze Python 2.6, neboť starší aplikace, které na serverech běží mají závislosti na knihovny, které pro verzi 2.6 nejsou dosupné, respektive dostatečně stabilní, aby obstály v ostrém komerčním provozu.

Byl mi tedy přidělen virtuální server se systémem CentOS release 5 Final s jádrem 2.6.32, na který byl nainstalován nejnovější verze Python 2.6 v přesné verzi Python 2.6.6. Na této verzi byla celá práce vyvíjena a bude na ní také nasazena v ostrém provozu, proto není garantována funkčnost na odlišných verzích. Je to jedna z mála funkčních Python verzí pro Net-SNMP balík.

3.2 Výběr Net-SNMP balíku

3.2.1 Net-SNMP 5.4

Na virtuální testovací stroj byl rovněž nainstalován Net-SNMP 5.4, jelikož s touto verzí komunikovaly všechny ostatní firemní aplikace napsané v jazyce Perl. Byla otestována základní funkčnost pomocí jednoduchého GET příkazu

```
import netsnmp
netsnmp.snmpget("sysDescr.0", Version=2, Community="public", DestHost="10.76.1.4")

('3Com Baseline Switch 2952-SFP Plus Software Version 5.20 Release 1101P09 \r\
    nCopyright (c) 2004-2010 3Com Corporation. All rights reserved.',)
```

Výpis 2: SNMP požadavek GET

Problém nastal v případě, kdy bylo třeba načíst více hodnot. Při zavolání metody `snmpwalk`, která se v Pythonu neprovedla a skončila s chybovým hlášením.

Na řadu přišlo další prohledávání internetu a zjišťování, kde může být problém. Informací nebylo mnoho, ale z toho mála, které se mi podařilo najít, bylo z internetových diskuzí patrné, že chyby vykazuje tentokrát balík Net-SNMP.

Následně jsem popis chyby našel společně s nahlášením chyby, tzv. Bugreportem číslo `581185` systému debian. Chyba označovala jako viníka balík `libsnmp-python`, který byl součástí Net-SNMP 5.4.2.1.

```
import netsnmp
netsnmp.snmpwalk("sysDescr", Version = 2, DestHost = "10.76.1.4", Community = "public")

error: walk: unknown python error (varlist)
```

Výpis 3: Chyba SNMPwalk u Net-SNMP 5.4.2.1

Začalo další hledání, zda-li lze na tento problém nějakým způsobem vyzrát. Přestože to již vypadalo beznadějně a namísto řešení se objevovaly další a další bugreporty v různých systémech, svitla naděje.

3.2.2 Net-SNMP 5.5

Mimo oficiální webové stránky projektu byl nalezen changelog, který popisoval aktualzaci Python modulu pro Net-SNMP. Ta měla být vyřešena panem Janem Šafránkem ze společnosti RedHat. Jednalo se o update Net-SNMP 5.4.2.1-6, který byl přenesen také do balíku, jenž se objevil také na oficiálním webu. Jeho verze byla Net-SNMP 5.5.1.

Byly staženy nové zdrojové soubory, ty zkmopilovány a nahrány na virtuální server. Testování mohlo pokračovat, alespoň se to tak zdálo, neboť SnmpWalk začal vracet správné výsledky.

```
import netsnmp
netsnmp.snmpwalk("sysDescr", Version = 2, Community = "public", DestHost = "10.76.1.4")

('3Com Baseline Switch 2952-SFP Plus Software Version 5.20 Release 1101P09 \r\
    nCopyright (c) 2004-2010 3Com Corporation. All rights reserved.',)
```

Výpis 4: Správný SNMPwalk u Net-SNMP 5.5.1

Po následném testování a zkoumání, zda-li vše končně funguje byla nalezena další chyba. Pokud byl vyvořen `session`, došlo k pádu celého interpretu Python a nebylo možné pokračovat.

```
import netsnmp
spojeni = netsnmp.Session(Version = 2, Community = "public", DestHost = "10.76.1.4")

0x080b7e2d in call_function (pp_stack=0xbffff448, oparg=2) at ../netsnmp/client.c:147
Program received signal SIGSEGV, Segmentation fault.
```

Výpis 5: Otevření SNMP sessionu

3.2.3 Net-SNMP 5.6

Naštěstí někoho napadlo zkontrolovat oficiální SVN projektu a najít testovací verze. Ty byly nalezeny, zkompilovány a funkční.

Poznámka 3.1 K dnešnímu dni je již oficiálně vydán balík Net-SNMP 5.6.1, který problémy řeší.

Problémy s Python knihovnami zmizely, avšak vznikly problémy s knihovnami pro Perl, který v aktuální kompilaci musel být zakázán. Lze předpokládat, že v dalších oficiálních verzích bude problém vyřešen.

Poznámka 3.2 Problémy se nevyskytují pouze v okamžiku, kdy přistupujeme ke knihovně správně. Jakmile například spustíme v Sessionu příkaz Walk s nesprávným parametrem (takový, který není typu netsnmp.VarList) dojde opět k ukončení interpretu s ozámením SEGFAULT.

```
import netsnmp
spojeni = netsnmp.Session(Version = 2, Community = "public", DestHost = "10.76.1.4")
spojeni.walk("system")
Segmentation fault
```

Výpis 6: Špatné volání SnmpWalk

Pro tuto práci je však Perl druhořadý, nebudeme se proto tímto problémem zabývat a ukážeme si správné volání metody `walk` uvnitř Sessionu.

```
import netsnmp
spojeni = netsnmp.Session(Version = 2, DestHost = "10.76.1.4", Community = "public")
var = netsnmp.Varbind("system")
vars = netsnmp.VarList(var)
spojeni.walk(vars)

('3Com Baseline Switch 2952 SFP Plus Software Version 5.20 Release 1101P09 \r\
    nCopyright (c) 2004 2010 3Com Corporation. All rights reserved.',
 '.1.3.6.1.4.1.43.1.8.73',
 '1014823748',
 '3Com Corporation.',
 'lbxovasw3',
 'Marlborough, MA 01752 USA',
 '78')
```

Výpis 7: Správné volání SnmpWalk

Takovéto volání má obrovskou nevýhodu. Knihovna `netsnmp` provádí `walk` tím způsobem, že pouze volá v cyklu SNMP příkaz GetNext. Pro malé uzly to můžeme zanedbat, ale vysvětlíme si, v čem je problém pro nasazení do našeho programu.

GetNext funguje tak, že posílá dotaz na jedinou hodnotu v MIB stromu. Když se dostaví očekávaný výsledek, zavolá se GetNext znovu a dostane další hodnotu v pořadí stromové struktůry. V našem případě potřebujeme načítat stovky údajů z například několika desítek zařízení. Pokud bychom použili nativní metodu Walk, může se nám stát, že zahltíme celou síť pouze dotazy na stav zařízení.

Bohužel Python knihovna Net-SNMP nenabízí funkci BulkWalk, která načítá data hromadně. Existuje zde pouze metoda GetBulk, jenž ale postrádá veškerou dokumentaci, respektive uveřejněná dokumentace je nefunkční. V době vzniku této bakalářské práce navíc nebyl nalezen žádný framework či příklad, který by daný problém řešil. Proto jsem po mnoha hodinách zkoušení napsal ukázkový skript na funkci BulkWalk, jež je uveřejněn a popsán v sekci Popis programu.

3.2.4 Net-SNMP 5.6.1

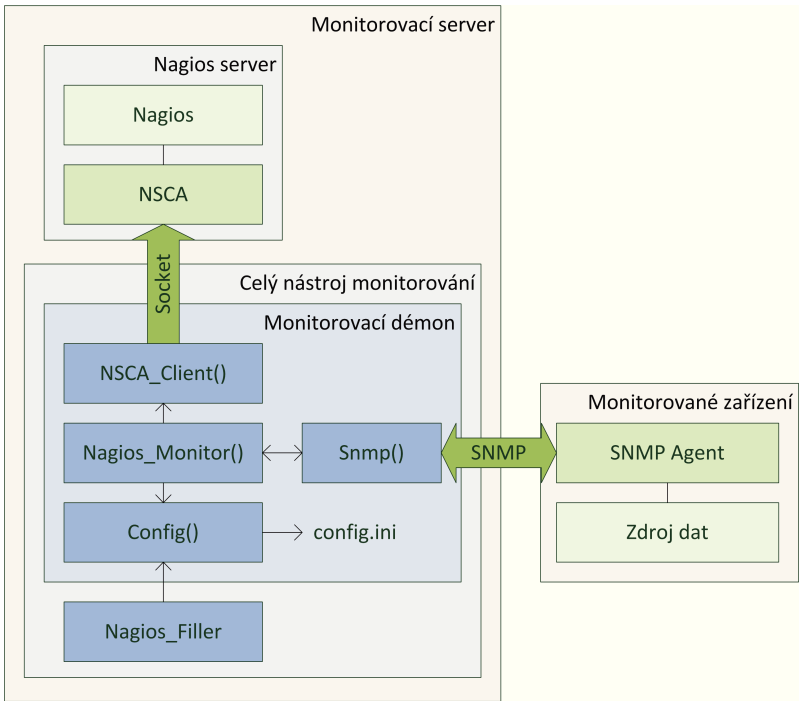
Na oficiálním webu http://www.net-snmp.org/ se ke dni 1. 4. 2011 uveřejnila nová verze Net-SNMP balíku. Byl testován na systému Ubuntu 10.04 LTS s jádrem 2.6.32-30 a Python 2.6.5. Testované příkazy a skripty nevykazovaly žádné chyby a veškeré výsledky byly správné.

Tento balík můžeme rovněž považovat za funkční a pro naši práci použitelný.

4 Popis programu

Celý program je rozdělen na několik částí, kdy každá se stará pouze o dílčí řešení problému. Tyto části jsou jednotlivé knihovny a každá z nich je připravena na univerzální použití mimo tento celek.

Jako hlavní část lze označit třídu `Nagios Monitor`. Ta je načítána do vláken a řeší základní funkcionalitu. Pomocí ostatních tříd si stáhne požadované informace, vyhodnotí je a odešle na server. Důležitou funkcí je, aby si knihovna pamatovala minulé testy, z důvodů výpočtu rozdílu mezi těmito testy.



Obrázek 3: Komunikace jednotlivých částí

Předpokládáme, že na monitorovacím serveru je spuštěn systém Nagios a NSCA server. Démon se připojí pomocí SNMP na testované zařízení. Hromadně získá potřebné informace, vyhodnotí je a výsledky odešle serveru NSCA. V drtivé většině použití bude sledovací démon umístěn na stejném fyzickém stroji, jako Nagios. Je však možno tuto část oddělit a provozovat ji v jiné části sítě.

4.1 SNMP knihovna

Bez nadsázky můžem říci, že SNMP knihovna byla nejproblematičtějším místem celé bakalářské práce. Téměř každý řádek musel být, pro svou absenci kvalitní a funkční dokumentace, psán stylem pokus-omyl.

V samotném konstruktoru se nastavují všechny atributy potřebné pro připojení. Jeho parametry jsou tedy verze protokolu, adresa zařízení a komunita. Obsahuje také následující metody

- **connect()** - naváže spojení se zařízením, které bylo předem specifikováno
- **is_tree(value, tree)** - vyhodnotí, zda-li dotazovaná hodnota `value` patří do aktuálního uzlu stromu `tree`
- **get_tree_oid(tree_name)** - načte OID hodnotu dotazovaného MIB stromu `tree_name`
- **walk(oid)** - projde rekurzivně celý MIB strom, začínající specifikovanou hodnotou `oid`

Příklad použití SNMP knihovny:

```
from Snmp import Snmp
moje_snmp = Snmp(version = 2, hostname = "localhost", community = "public")
moje_snmp.connect()
vysledek = moje_snmp.walk("system")
print vysledek

[['localhost', ['sysDescr'], '0', 'Linux johnson-laptop 2.6.32-30-generic #59-Ubuntu SMP
    Tue Mar 1 21:30:46 UTC 2011 x86_64', 'OCTETSTR'],
 ['localhost', ['sysObjectID'], '0', '.1.3.6.1.4.1.8072.3.2.10', 'OBJECTID'],
 ['localhost', ['sysUpTime', 'sysUpTimeInstance'], '', '101922666', 'TICKS'],
 ['localhost', ['sysContact'], '0', 'bed189@vsb.cz', 'OCTETSTR'],
 ['localhost', ['sysName'], '0', 'johnson-laptop', 'OCTETSTR'],
 ['localhost', ['sysLocation'], '0', 'Vsude mozne', 'OCTETSTR']]
```

Výpis 8: Příklad užití knihovny SNMP

Z ukázky je patrné, že výstupem metody `walk` je seznam jednotlivých výsledků. Hodnoty jsou uloženy v pořadí

Hostname	MIB cesta	ID uzlu	Hodnota	Typ hodnoty

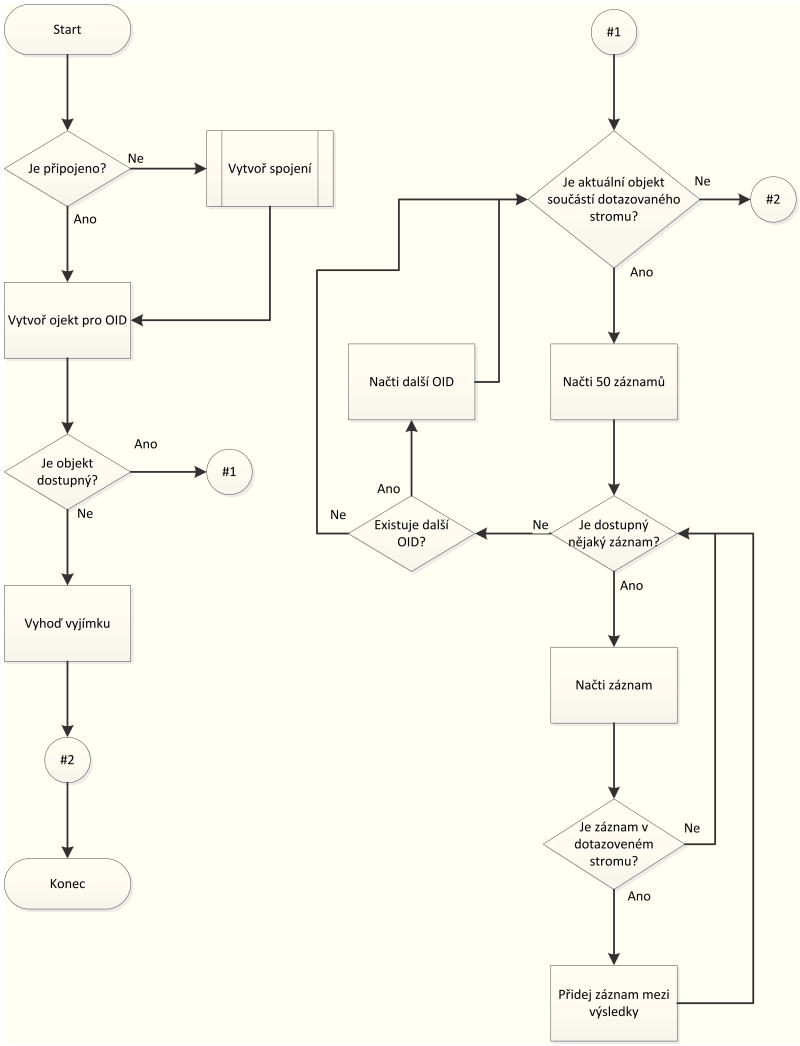
Tabulka 5: Popis výsledků metody walk

Význam těchto hodnot je následující

1. **Hostname** - adresa dotazovaného zařízení. Může se jednat o IP adresu, nebo platný DNS záznam
2. **MIB cesta** - relativní cesta mezi kořenem dotazovaného stromu a aktuálním prvkem. Jednotlivé úrovně jsou seřazeny v seznamu
3. **ID uzlu** - pořadové číslo, specifikující hodnotu řádku v SNMP tabulce. Je shodné s posledním číslem OID hodnoty, například pořadové číslo rozhraní v uzlu `ifIndex`
4. **Hodnota** - hodnota navráceného výsledku
5. **Typ hodnoty** - určuje, jakého typu SNMP objektu je vrácená hodnota. Nabývá hodnot popsaných dříve.

4.1.1 SNMP BulkWalk v Pythonu

V průběhu tvorby práce bylo zjištěno, že neexistuje žádný návod na hromadné načítání hodnot prostřednictvím Net-SNMP. Na internetu se sice objevují skripty a frameworky, které problém zdnánlivě řeší, žádný z nich však není dostatečně funkční. Buďto načítají hodnoty jednu po druhé funkcí getNext, nebo užívají funkce getBulk, avšak nepočítají s případem, že v dotazovaném stromu bude nalezen podstrom. Takováto řešení vrací nekorektní výsledky, v případě že se jim podaří informace zjistit. V opačném případě skončí chybou.



Obrázek 4: Vývojový diagram pro funkci BulkWalk

Budeme-li chtít tedy popsat přístup k Net-SNMP funkci `getbulk`, bude to vypadat následovně

```
getbulk(offset, count, vars)
```

- **offset** - udává počet hodnot, které se mají vynechat v pořadí od zadaného OID (výchozí hodnotu doporučuji volit 0)
- **count** - počet hodnot, které se mají načíst najednou (doporučuji hodnotu 50, neboť s vyššími docházelo k nekorektnímu načítání stromu a nižší mají za následek příliš mnoho dotazů)
- **vars** - určuje OID, jenž se vezme jako výchozí a načítají se následující hodnoty. OID je obaleno dvěma kontejnery a nastavuje se jako `VarList(Varbind(OID))`. Do tohoto atributu `vars` se následně uloží všechny výsledky, jako seznam typu `netsnmp.VarList`, který obsahuje hodnoty `netsnmp.Varbind`

`VarList` tedy udává seznam položek `Varbind`, jenž obsahuje následující atributy:

- **tag** - OID adresa
- **iid** - ID položky
- **type** - typ SNMP objektu
- **val** - navrácená hodnota

Z tohoto důvodu jsem proto do práce přidal skript `snmp_walk.py`, který je ekvivalentem funkce BulkWalk, přístupné z příkazové řádky.

Budeme-li chtít skript spustit, použijeme následující syntaxi

```
pouziti: snmp_walk.py <prepinac> <snmp_node>

    prepinace:
    -h    hostname
    -c    community
    -v    snmp_version
```

Pro úplnost bych zde uvedl nejdůležitější část skriptu, kterou je metoda `get_walk`

```
def get_walk(self):
    #vytvoří spojení
    self._ses = netsnmp.Session(DestHost = self.hostname, Version = self.version, Community =
        self.community)
    #příprava seznamu pro výsledky
    self.results = []
    #priprava seznamu pozadovanych oid
    vars = netsnmp.VarList(netsnmp.Varbind(self.oid))
    #konec pasaze pro walk
    self._ses.UseLongNames = True
    #otestuje spojeni a zjsiti aktualni strom
    self.tree = thistree = self.get_tree_oid(self.oid)
    #pokud bylo zadáno chybné OID, vyhodíme vyjímku
    if thistree == -1:
        print "Chyba: Nebylo zadáno platné OID"
        return -1
```

```
#dokud se nacházíme v aktualním stromě
while(self.is_tree(thistree)):
    #načteme dalších 50 hodnot
    result = self._ses.getbulk(0,50,vars)
    #projdeme všechny vrácené výsledky
    for item in vars:
        if self.is_tree(item.tag):
            #pokud se výsledek nalézá stále v požadovaném stromu, přidáme výsledek k ostatím
            self.results.append([self.hostname, item.tag[len(thistree)+1:].split("."), item.iid,
                item.val, item.type])
        else:
            #pokud jsme se ocitli mimo požadovaný strom, ukončíme průchod
            break

    if not vars[-1].iid:
        break
    else:
        #v případě, že poslední výsledek má OID, nastavíme jej jako výchozí pro nové čtení
        vars = netsnmp.VarList(netsnmp.Varbind(vars[-1].tag,vars[-1].iid))

#vrátíme požadované výsledky
return self.results
```

Výpis 9: Funkční metoda pro BulkWalk

4.2 NSCA knihovna

Jelikož protokol Nagios NSCA není v Pythonu dostupný, bylo třeba naprogramovat nového klienta tohoto protokolu.

Bohužel nebyla nalezena žádná dokumentace, která by přesně popisovala komunikaci a chování mezi NSCA serverem a klientem. Vycházel jsem tedy z oficiálního `send_nsca.pl` skriptu. Jedná se o klientskou stranu NSCA, uveřejněnou na webu http://exchange.nagios.org/. Následně byl Perl skript přepsán do jazyka Python, což navíc vyžadovalo nastudování syntaxí jazyka Perl a jeho užití.

Konstruktor definuje IP adresu a port Nagios serveru. Volitelně lze nastavit verzi NSCA paketu, ta je ve výchozím stavu verze 3.

Knihovna obsahuje tyto metody:

- **pack_data(packet_version, crc, timestamp, return_code, hostname, service, status)** - zpracuje data do tvaru paketu. Vstupními atributy jsou `packet_version` udávající vezi paketu, kontrolní součet `crc`, časovou značku `timestamp`, návratový kód Nagiosu `return_code`, název zařízení `hostname`, název služby `service` a textový `status` výsledku. Jako výstup je vytvořen NSCA paket požadovaného tvaru
- **add_data(new_data)** - přidá další výsledky(`new_data`), mezi ostatní, určené k odeslání

- **myxor(xor_key, str_data)** - zašifruje metodou XOR `str_data` podle vlastního `xor_key` klíče
- **generate_crc32_table()** - vytvoří vlastní tabulku, podle které se budou generovat 32bitové kontrolní součty
- **calculate_crc32(buffer)** - pro hodnotu `buffer` vytvoří 32bitový kontrolní součet
- **connect()** - připojí se k NSCA serveru a odešle vložené výsledky

```
from nsca_client import NSCA_Client
nsca = NSCA_Client(nagios_ip, nagios_port)
nsca.add_data(vysledek_testu)
nsca.connect()
```

Výpis 10: Příklad užití knihovny NSCA

4.3 Konfigurační knihovna

Pro právnou funkčnost je třeba definovat, co, kde, a jak se má monitorovat a kam se to bude na konci odesílat.

Jako úložiště pro konfiguraci byl zvolen, kvůli své jednoduchosti a přehlednosti, INI soubor. Pro tyto soubory se v Pythonu využívá třídy `ConfigParser`.

Abychom si načítání hodnot co nejvíce zjednodušili a zpřehlednili, bylo zapotřebí naprogramovat novou třídu, která by se o načítání nastavení starala. Její výhodou je například také to, že pokud se v průběhu užívání celého projektu rozhodne administrátor ukládat konfigurace do databáze či jiné formy, dojde k přepsání pouze této třídy. Zbytek programu je tedy nezávislý na formě konfigurace.

Třída `Config` má jediný parametr konstruktoru, kterým je název souboru s uloženou konfigurací. Obsahuje také další metody pro načítání informací z tohoto souboru:

- **get_globals()** - vrátí všechny globální informace ve formě slovníku
- **get_global(key)** - vrátí hodnotu globální informace `key`
- **get_all()** - vrátí všechny položky nastavení jako slovník
- **get_nagios_server()** - vrací slovník, obsahující informace o Nagios serveru. Klíči slovníku jsou `host` a `port`
- **get_devices()** - vrátí informace o všech zařízeních v konfiguraci
- **get_deviceinfo(device, key)** - vrátí hodnotu klíče `key`, nastavenou u zařízení `device`
- **get_value(section, key)** - vrátí hodnotu uloženou v sekci `section`, pod klíčem `key`

```
from config import Config
conf = Config("soubor.ini")
serv = conf.get_nagios_server()
info = conf.get_devicienfo("sw1","ip")
print serv
{'host': '127.0.0.1', 'port': 1234}
print info
'10.76.1.4'
```

Výpis 11: Příklad užití knihovny Config

```
; globalni cast
[global]
nagios-host=10.153.31.1 ; adresa nagios serveru
nagios-port=5667        ; port nagios serveru
check_interval=60       ; interval kontroly
pkterr_yellow=0         ; hodnota warning
pkterr_red=0            ; hodnota critical

; definice zarizeni
[sw1]
ip=127.0.0.1            ; adresa zarizeni
version=2               ; verze snmp protokolu
community=public        ; komunita snmp
```

Výpis 12: Příklad konfiguračního souboru

4.4 Nagios_Monitor

Třída `Nagios_Monitor` se stará o získávání informací ze zařízení, pomocí knihovny `Snmp()`, vyhodnotí a zpracuje všechny potřebné hodnoty a následně je odešle serveru Nagios, prostřednictvím knihovny `NSCA_Client()`.

Její ulohou je tedy získat správné hodnoty, porovnat je s hodnotami posledního předešlého měření, získat rozdíl těchto hodnot a podle konfigurace rozhodnout, zda-li se jedná o chybu a výsledky odeslat systému Nagios v požadovaném tvaru.

Třída dedí z `Thread`, proto umožnuje použití vláken. Ty jsou důležitá pro paralelní vykonávání testů. Pokud by nebylo užito vláken, mohlo by se jednoduše stát, že z důvodu získávání velkého množství informací z mnoha zařízení by docházelo k neúměrnému prodlužování testovaných intervalů v chaotické okamžiky.

Mimo metody `run`, starající se o získávání dat a vyhodnocování testu obsahuje třída i další metody:

- **reset_thread()** - inicializuje nové vlákno
- **set_nagios(ip, port)** - nstavuje údaje (`ip` adresu a `port`) o Nagios serveru pro odesílání dat
- **set_device(ip, version=2, community="public")** - nastavuje `ip` adresu, verzi SNMP protokolu `version` a SNMP komunitu `community` dotazovaného zařízení

- **make_diff()** - projde výsledek testu, nalezne hodnoty, fungující jako čítač, a vytvoří jejich rozdíl
- **serch_in_archive(host, oid, id)** - vyhledá v milulém měření hodnoty odpovídající IP adrese `host`, cestě `oid` na pozici `id`
- **search_in_list(my_list, oid, id=0, host=None)** - vyhledá v seznamu `my_list` hodnotu `oid` s pořadovým číslem `id`, patřící IP adrese `host`. V případě, že není `host` uveden, bude se vyhledávat podle prvního nalezeného
- **get_tree(my_list, oid, host=None)** - prohledá seznam výsledků `my_list` a vrátí všechny záznamy patřící do stromu `oid` a IP adrese `host`. Pokud není `host` zadán, opět se porovnává podle prvního nalezeného
- **send_to_nagios()** - odešle vyhodnocená data Nagios serveru

Ukázkový příklad demonstruje spuštění dvou paralelních testů na zařízení s ip adresami 10.76.1.4 a 10.76.2.6.

```
from nagios_monitor import Nagios_Monitor
test1 = Nagios_Monitor()
test1.set_nagios("127.0.0.1",1234)
test1.set_device("10.76.1.4")
test2 = Nagios_Monitor()
test2.set_nagios("127.0.0.1",1234)
test2.set_device("10.76.2.6")
test1.start()
test2.start()
test1.join()
test2.join()
```

Výpis 13: Příklad užití knihovny Nagios Monitor

Soubor `nagios_monitor.py` slouží zároveň jako hlavní spouštěcí skript celého démona.

Po spuštení načte hodnoty z konfiguračního souboru, využívajíce knihovny `Config()`, vytvoří vlákna pro testování a v konfiguračním souboru předem definovaných inervalech provádí testování předepsaných zařízení a odesílá informace systému Nagios.

4.5 Nagios Filler

Při zadání práce bylo podmínkou nejen vytvořit nástroj k testování, ale také skript, který by byl schopen z konfiguračního souboru vyčíst všechny informace a vytvořit definici pro přidání nových zařízení a služeb do systému Nagios.

Byl proto vytvořen skript `nagios_filler.py`, plnící tuto úlohu. Jako parametr při spuštění se zadá umístění konfiguračního souboru. Skript si následně z tohoto souboru načte informace o zařízeních a vytvoří instanci třídy `Snmp()`, pomocí které načte ze zmíněných zařízení všechny potřebné údaje. Jakmile dokončí sběr informací, vygeneruje na standardní výstup definici nastavení potřebného pro Nagios.

Tuto definici následně administrátor ručně vloží do Nagiosu. Ten provede aktualizaci a akceptuje nové služby a zařízení. Od téte chvíle bude Nagios příjmat výsledky nových zařízení, definovaných v konfiguračním souboru démonu.

5 Závěr

Výsledkem této práce je plně funkční nástroj pro sběr dat ze síťových zařízení, analýzu těchto dat a následné odevzdání vyhodnocených výsledků na server se systémem Nagios, shromažďující tyto výsledky. Nástroj funguje jako démon a plně nahrazuje dosavadní LXMON, sloužící pro toto monitorování.

Pro realizaci bylo zapotřebí prozkoumat dostupné možnosti síťové komunikace pomocí protokolu SNMP, který bylo potřeba nastudovat. Byl vyvinut postup a návod na komunikaci s Net-SNMP knihovnou pomocí jazyka Python a následné užití této knihovny k hromadnému načítání dat. To by mělo posloužit jako odrazový můstek pro případné další studenty, kteří budou řešit tyto funkční problémy.

Realizace této bakalářské práce pro mne byla přínosná zejména z důvodů detailního pochopení SNMP komunikace a řešení nezvyklých problémů s jazykem Python a jeho knihoven.

Do případné další verze by bylo možné upravení knihovny pro komunikaci s SNMP a přidání podpory protokolu verze 3. Vhodné by taktéž bylo rozšířit možnosti šifrování odchozích dat o bezpečnější a sofistikovanější algoritmy, které by ve spojení s novějším SNMP protokolem zajistily dostatečně bezpečný přenos dat.

Jan Bednář

6 Reference

[1] ŠVEC, Jan, *Seriál Python - Létající cirkus* Root.cz : informace nejen ze světa Linuxu [online]. Verze 1.0. 2001 [cit. 2011-04-23]. Létající cirkus. Dostupné z WWW: http://www.root.cz/clanky/letajici-cirkus/

[2] JAMRICH, Marián, *Nagios: monitorovanie počítačovej siete* Root.cz : informace nejen ze světa Linuxu [online]. 2010-12-15 [cit. 2011-05-03]. Dostupné z WWW: http://www.root.cz/clanky/nsca-nrpe-a-vlastne-nagios-pluginy/

[3] OSTERLOH, Heather, *TCP/IP : Kompletní průvodce.* [s.l.] : SoftPress s.r.o., 2003. 512 s. ISBN 80-86497-34-8.

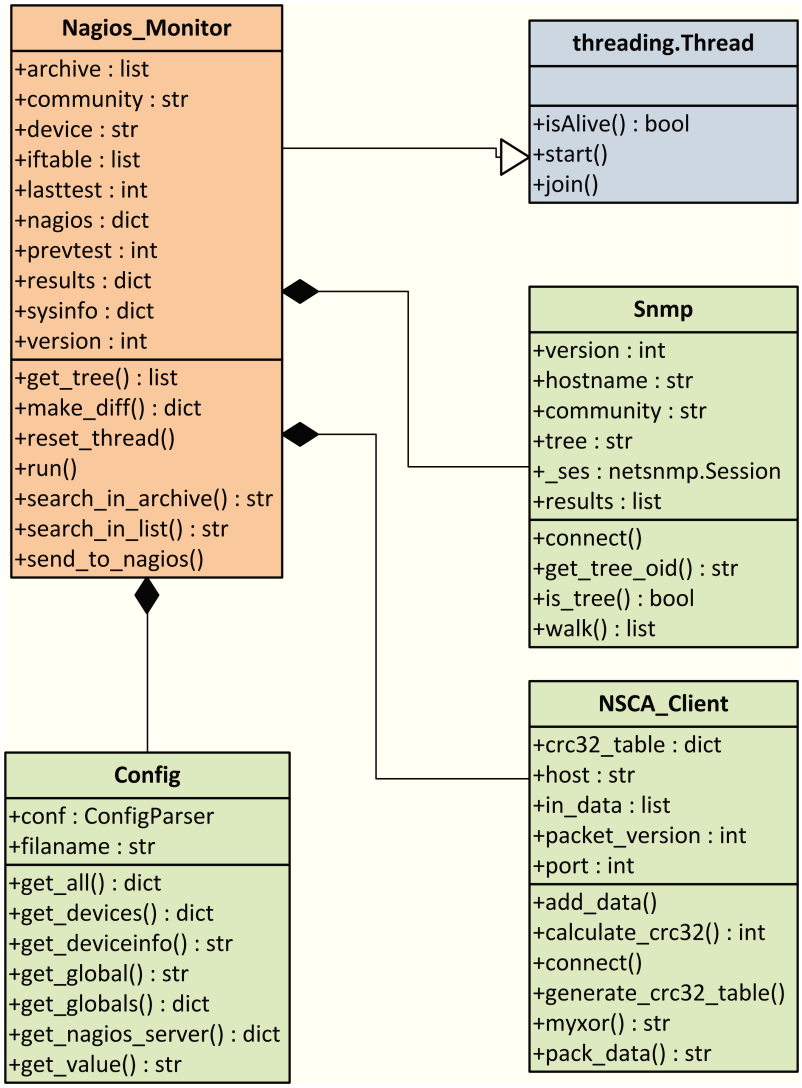
[4] BRUEY, Douglas, *SNMP: Simple? Network Management Protocol.* RaneCorporation : RaneNote [online]. 161. 2005 [cit. 2011-04-23]. Dostupné z WWW: http://www.rane.com/note161.html rane-note

[5] M. KOZIEROK, Charles, *The TCP/IP Guide - SNMP Version 2 (SNMPv2) Message Formats.* TCP/IP Guide : TCP/IP Reference You Can Understand [online]. Version 3.0. 2005-11-25 [cit. 2011-04-23]. Dostupné z WWW: http://www.tcpipguide.com/free/t_SNMPVersion2SNMPv2MessageFormats-5.htm

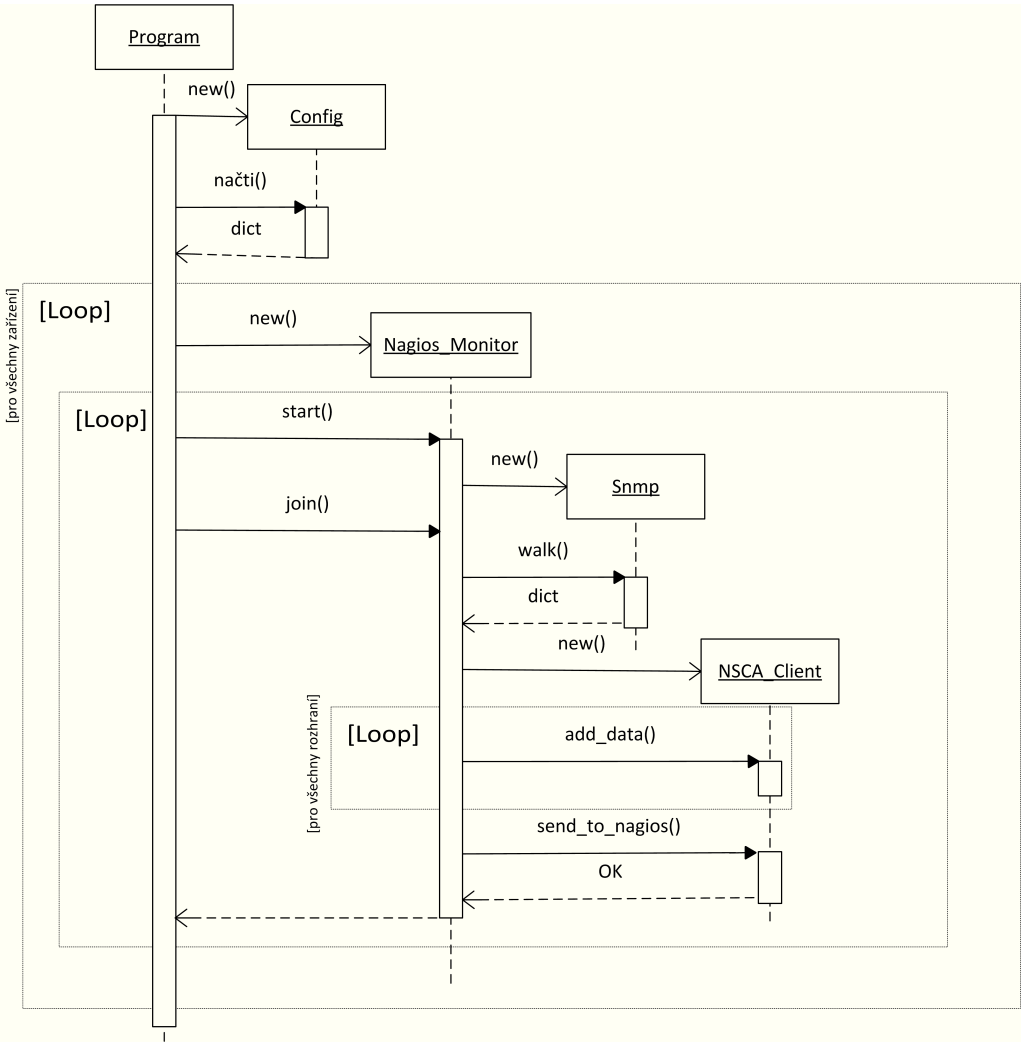
[6] BOUŠKA, Petr, *SNMP - Simple Network Management Protocol.* Samuraj-cz [online]. První vydání. 2006-12-20 [cit. 2011-04-23]. Dostupné z WWW: http://www.samuraj-cz.com/clanek/snmp-simple-network-management-protocol/

A Diagramy

V této části se nachází diagramy popisující chování programu jako celku. Kvůli jejich obecnosti a velikosti byl přesunuty do speciální části v přílohách.



Obrázek 5: Třídní UML diagram



Obrázek 6: Sekvenční diagram běhu démona